

Day 5

1st NextGen Hackathon



NextGen
Next Generation Triggers

Events generators on GPU

Daniele Massaro



NexTGen
Next Generation Triggers

Sum-up of the week

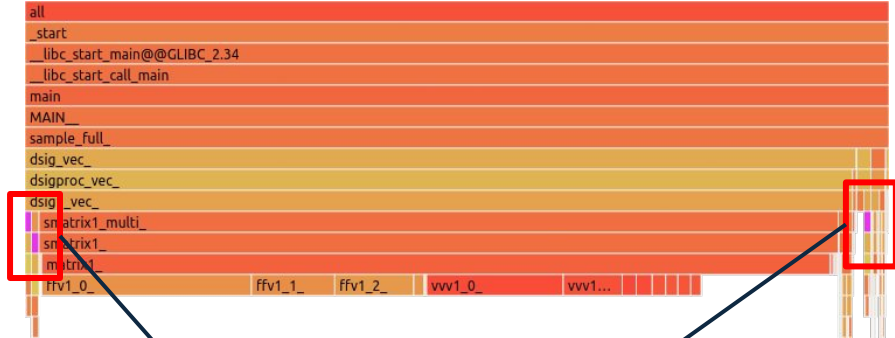
- Learn what is MadGraph and how to run it.
- Learn how to install the plugin to run the GPU porting of MadGraph.
- Learn about Adaptyst profiler and hands-on.
- Profile MadGraph and produce several flamegraphs:
 - appreciate the speedup of the GPU version with respect to the original FORTRAN version;
 - study next steps in the development:
 - IO (aka writing of the LHE event file) can be improved if using a binary (like H5) format;
 - PDF interpolation and evaluation can be done through the LHAPDF library, which has a GPU version;
 - **(today)** there seems to be an issue with the PDF caching mechanism implemented in MadGraph.

PDF caching

- The GPU plugin of MadGraph modifies slightly the subprocess structure to make it more suitable for hardware acceleration:
 - the “original” FORTRAN implementation has been slightly rewritten to accommodate this change;
 - this new implementation should be equivalent to the original one

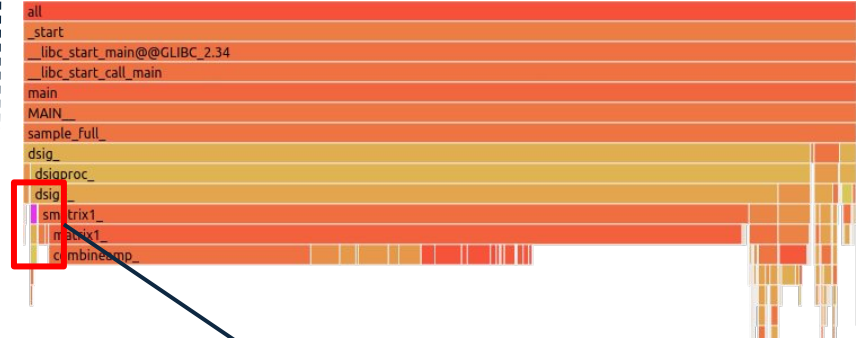
PDF caching

New



3 calls to the PDF routines

Original



1 call to the PDF routines
(the other calls are cached)

Maybe there is something to fix in the cache?

Torch Inference in Heterogenous CMS Software

Leonardo Beltrame, Lukasz Michalski, Felice Pantaleo,
Davide Valsecchi, Christine Zeh



NexTGen
Next Generation Triggers

Where we started

- Already had a working solution with:



- Just-in-time (JIT) model support + custom execution chain in CMS software with heterogeneous mindset
- Interface for direct conversion of SoA data into PyTorch tensors without explicit data movement. Automatic computation of tensor strides, and flexible column/tensor list ordering.

- But:



- No in depth profiling yet, only verification of correctness in some simple use cases.
- Limited understanding of bottlenecks
- Not yet structured for easy extension or maintenance

Plan for the week

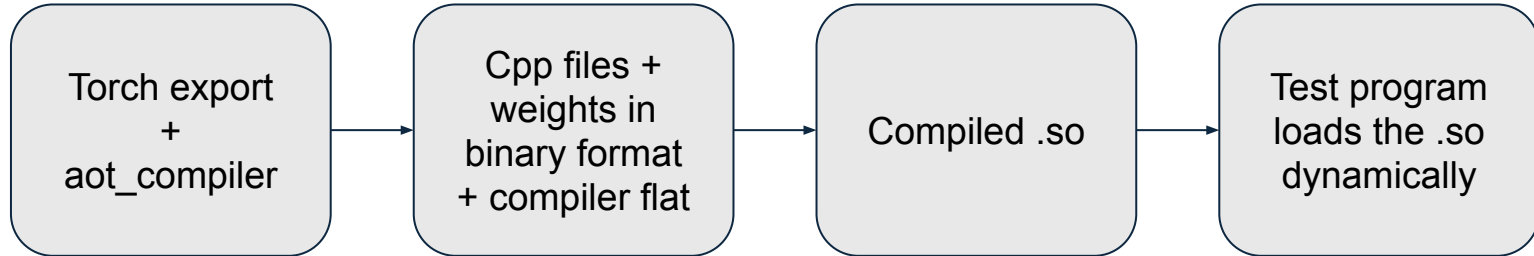
- **Integrate Torch-Alpaka Inference** in cmssw, open PR to collect feedback
- **Improve interface for SoA to torch::Tensor** improvements possible with merged PR: <https://github.com/cms-sw/cmssw/pull/47306>
- **Profiling & Optimization:**
 - Ensure compatibility with CMSSW's execution model.
 - Ensure we are able to lock all Torch internal optimizations (multithreading, Alpaka GPU queues).
- **Explore Ahead-of-Time (AOT) Compilation:** Investigate how AOT can improve model execution instead of relying on just-in-time (JIT) compilation.



Ahead-of-Time (AOT) Compilation



- Working example:
 - Add support for AOT model and run it from alpaka producer
 - Although we benefit from countless stuff we need to think about CMS deployment context.



Found a flag to export the sources instead of compiling directly!

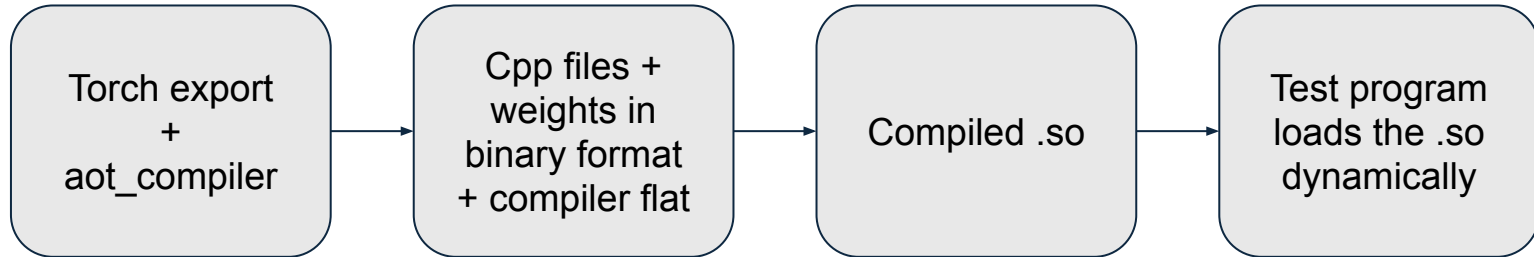


Compile it directly in the CMSSW environment (where libtorch is available)

Ahead-of-Time (AOT) Compilation



- What to investigate in following week:
 - Benchmark JIT vs AOT
 - Need the smart way of compiling and maintaining models for all architectures (we are tightly coupled with architecture)



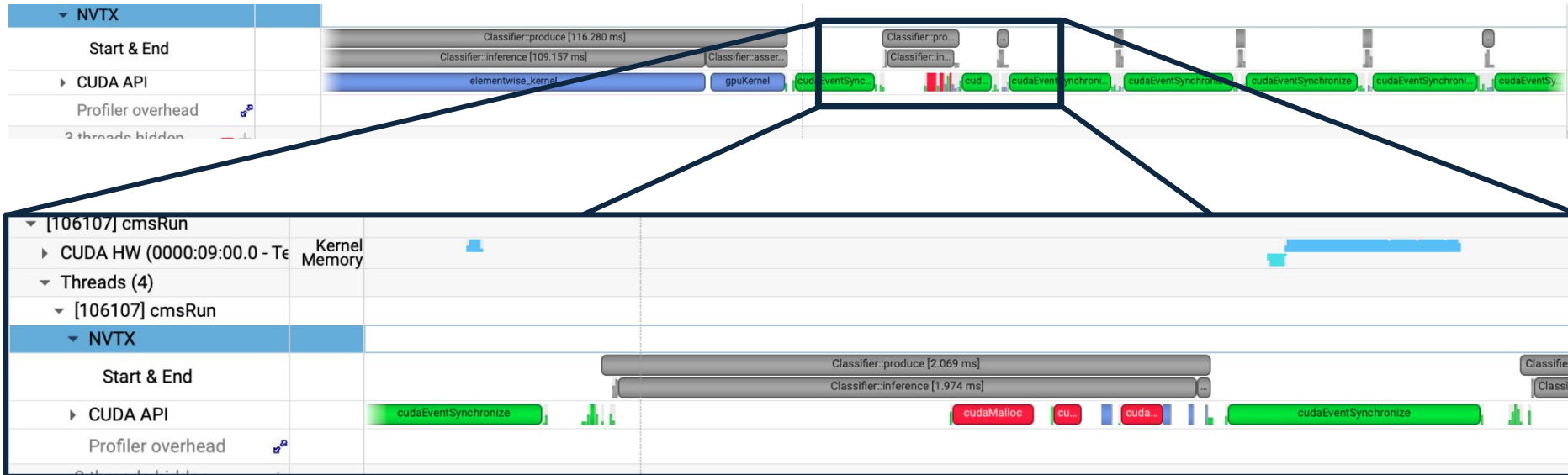
Found a flag to export the sources instead of compiling directly!



Compile it directly in the CMSSW environment (where libtorch is available)

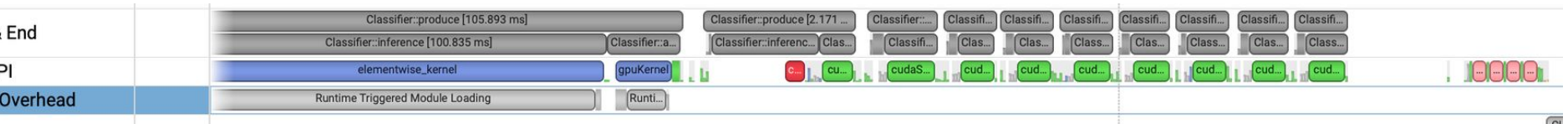
CMS software aware resources management

- Torch by default uses own caching allocator (it is not bad)
 - But we can see some explicit cudaMallocs that we might optimize



CMS software aware resources management

- Torch by default uses own caching allocator (it is not bad)
 - at the same time the library expose interface that enable us to plug our custom allocator, so we can take responsibility and control over memory allocation
 - <https://github.com/cms-sw/cmssw/blob/master/HeterogeneousCore/AlpakaInterface/interface/CachingAllocator.h>



CMS software aware resources management

- Torch by default uses own caching allocator (it is not bad)
 - So for single-thread single-stream the concept is working
 - The problem arises when we want to use multiple threads / streams in cms
 - Torch require one object that handle all cached memory related management, which cannot be switched every event

- With Felice's help, come up with the initial concept of the solution to use AlpakaService for device- and stream-aware caching allocator functions (malloc/free) for torch code
 - Need now to implement it correctly.

Future Plans

- Polish and integrate **custom allocator**
- Fully CMSSW-aware **memory/resource handling**
- Automate **AOT deployment** across architectures
- Restructure codebase for modularity and maintainability

Fast ML Inference with hls4ml

Dimitrios Danopoulos, Roope Niemi, Vladimir Loncar, Gaspard Le Gouic, Michael Divià, Quentin Berthet



NextGen
Next Generation Triggers

Results

FCN : Fully Convolutional Network for Jet reconstruction

- Initialized 2 trial synthesis for the pytorch version of the model (still running)
- Reduced issues of the Brevitas version to a channel order issue for resize and average pool
- Try PQuant

Results

NGT WP2.2: Implementation of classification model for L0 MDT Trigger

- Fully parallel NN implementation synthesized for FPGA, low precision with acceptable accuracy
- Few number of params, yet analysis showed additional resource savings could be achieved through pruning

fAD: flow anomaly detection for triggers

- Regular densely-connected NN model fully parallelized (75ns latency), custom precision
- Accuracy slightly reduced compared to the full-precision baseline, within acceptable margins for the target use-case.

Future Plans

Expand support for PQuant, facilitate the interaction with hls4ml

Future improvements for multi-graph flow in hls4ml, enable support for parallel synthesis of NN models
Planning to merge the feature into the main repo soon.

Identified major blockers with hls4ml that currently limit the ease of use
We will prioritize features that are needed by community

```
name: soa
on: workflow_dispatch
jobs: test_soa: runs_on: cern-nextgen-h100
```

uses: Jolly Chen, Leonardo Beltrame, Oliver Rietmann,
Simone Balducci, Simone Rossi Tisbeni,
Luca Ferragina, Aurora Perego, Davide Gadioli



NexTGen
Next Generation Triggers

Sum-up of the week

- Spent 1 day for each implementation (Reflection, Macros, Templates)
 - Lots of feedback for future work: iterators, jagged vectors, splice/subviews
- Created a shared benchmark for the three implementation
 - Created self contained execution environment
 - **Working GitHub action!!**
<https://github.com/cern-nextgen/wp1.7-soa-benchmark/actions/runs/14404957143/job/40399151015>
 - Working on automatically plotting results for 10 to 1000000 members
 - No fancy graphs yet :(
- Implement association map with the three different SoA versions

```

template <typename V, typename Score>
struct Indexes {
    uint32_t& indexes;
    V& values;
    Score& scores;
};

template <typename V, typename Score>
class AssociationMap {
public:
    AssociationMap(uint32_t nassociations, uint32_t nbins)
        : m_content(nassociations),
          m_offsets(nbins + 1),
          m_associations(nassociations),
          m_nbins(nbins) {}

    void initialize(const std::vector<uint32_t>& associations) {
        assert(m_offsets.size() == associations.size() + 1);
        m_offsets[0] = 0u;
        std::inclusive_scan(associations.begin(), associations.end(),
                           m_offsets.begin() + 1);
    }

    void fill(const std::vector<uint32_t>& associations,
              const std::vector<uint32_t>& indexes,
              const std::vector<V>& values, const std::vector<Score>& scores) {
        initialize(associations);
        auto temp_offsets = m_offsets;
        for (auto i = 0u; i < indexes.size(); ++i) {
            const auto position = temp_offsets[associations[i]]++;
            m_content.indexes[position] = indexes[i];
            m_content.values[position] = values[i];
            m_content.scores[position] = scores[i];
        }
    }

private:
    rmp::vector<Indexes<V, Score>> m_content;
    std::vector<uint32_t> m_offsets;
    uint32_t m_nassociations;
    uint32_t m_nbins;
};

```

The goal was to implement an association map using the different SoA libraries

- Jolly's library was the easiest library to use
- Easy construction of the SoA
- Of course, still dependent on an experimental compiler
- Very promising for future applications once C++26 (or C++29) is available

```

template <template <typename> typename F> struct Indexes {
    F<uint32_t> indexes;
    F<float> values;
    F<double> scores;
};

class AssociationMap {
public:
    AssociationMap(uint32_t nassociations, uint32_t nbins)
        : m_content{factory::default_wrapper<std::vector, Indexes,
                    wrapper::layout::soa>{
            nassociations}},
          m_offsets(nbins + 1), m_nassociations(nassociations), m_nbins(nbins) {}

    void initialize(const std::vector<uint32_t> &associations) {
        m_offsets[0] = 0u;

        std::vector<uint32_t> binsizes(m_nbins, 0);
        std::for_each(associations.begin(), associations.end(),
            [&binsizes](auto bin) -> void { ++binsizes[bin]; });
        std::inclusive_scan(binsizes.begin(), binsizes.end(),
            m_offsets.begin() + 1);
    }

    void fill(const std::vector<uint32_t> &associations,
              const std::vector<uint32_t> &indexes,
              const std::vector<float> &values,
              const std::vector<Score> &scores) {
        initialize(associations);
        auto temp_offsets = m_offsets;
        for (auto i = 0u; i < indexes.size(); ++i) {
            const auto position = temp_offsets[associations[i]]++;
            m_content[position].indexes = indexes[i];
            m_content[position].values = values[i];
            m_content[position].scores = scores[i];
        }
    }

private:
    wrapper::wrapper<std::vector, Indexes, wrapper::layout::soa> m_content;
    std::vector<uint32_t> m_offsets;
    uint32_t m_nassociations;
    uint32_t m_nbins;
}

```

- The SoA/AoS is relatively easy to use
- The use of templates makes it a bit verbose
- No constructor for the wrapper class.
Construction with a factory function
- The skeleton class takes only the container as
template parameter, so the types in the
columns need to be hard-coded

Testing compression of SoA

- Benchmarked the compression of SoA vs AoS in CMSSW
- Used the MACRO version of SoA since it's already available in CMSSW
- Compared AoS with “manual” SoA and with the MACRO generated SoA

```
struct SimplePhysicsObj2 {  
    double x;  
    double y;  
    float z;  
};  
using AoSView2 = std::vector<SimplePhysicsObj2>;
```

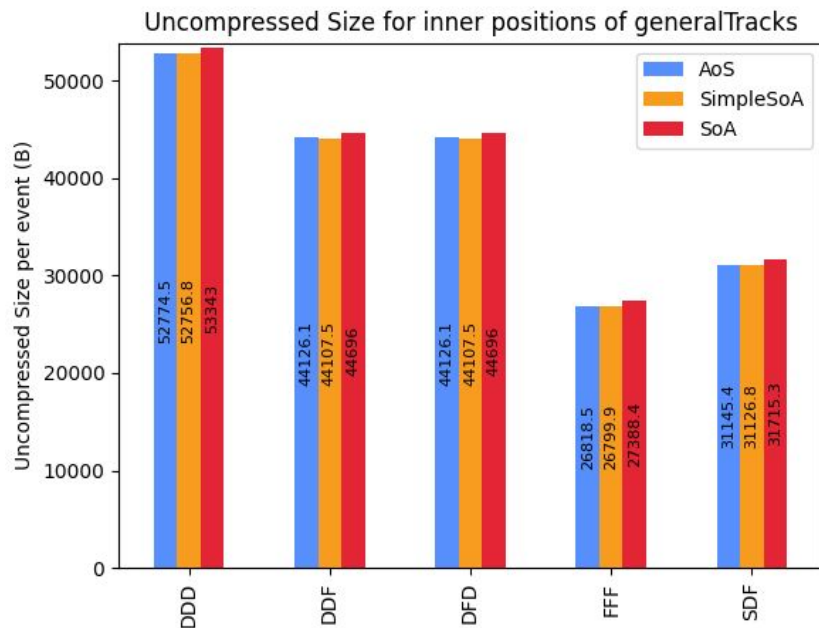
```
GENERATE_SOA_LAYOUT(PhysicsObj2Template,  
                    SOA_COLUMN(double, x),  
                    SOA_COLUMN(double, y),  
                    SOA_COLUMN(float, z))  
using PhysicsObj2 = PhysicsObj2Template<>;  
using PhysicsObj2View = PhysicsObj2::View;
```

```
struct SimpleSoAPhysicsObj2 {  
    SimpleSoAPhysicsObj2() = default;  
    SimpleSoAPhysicsObj2 (size_t elems){  
        x.reserve(elems);  
        y.reserve(elems);  
        z.reserve(elems);  
    }  
    std::vector<double> x;  
    std::vector<double> y;  
    std::vector<float> z;  
};
```

<https://srossiti.web.cern.ch/NGTHackathon/plot/>

Testing compression of SoA (contd.)

DDD = double, double, double
DDF = double, double, float
DFD = double, float, double
FFF = float, float, float
SDF = short, double, float

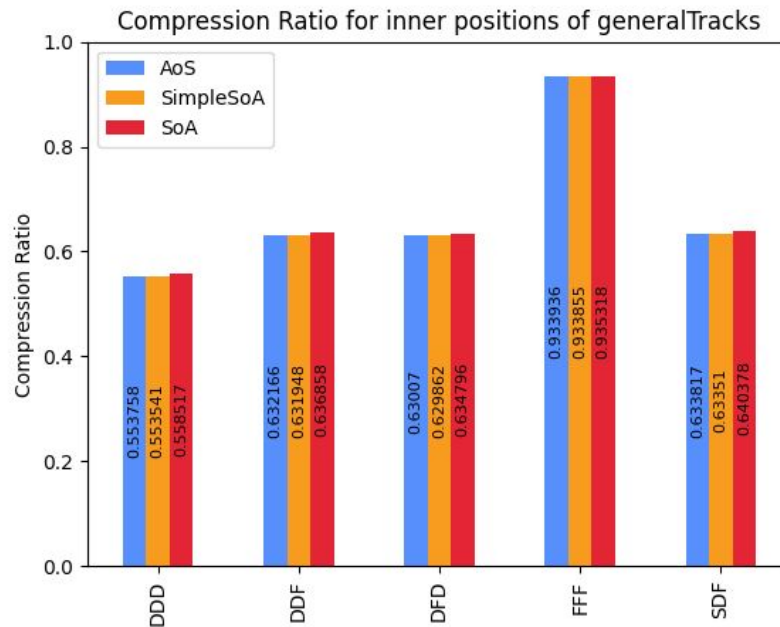
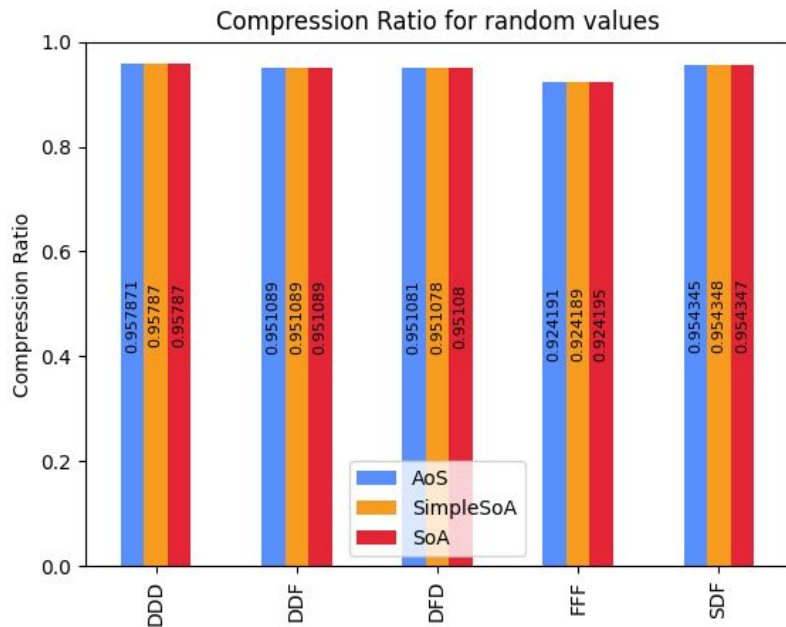


<https://srossiti.web.cern.ch/NGTHackathon/plot/>

Testing compression of SoA (contd.)

Compression ratio = compressed / uncompressed (lower is better)

DDD = double, double, double
DDF = double, double, float
DFD = double, float, double
FFF = float, float, float
SDF = short, double, float



<https://srossiti.web.cern.ch/NGTHackathon/plot/>

Fast ML Inference

Alessandro Crespi, Jamie Gooding, Enrico Lupi,
Lorenzo Moneta, Anastasiia Petrovych,
Sanjiban Sengupta, Jonas Rembser



NexTGen
Next Generation Triggers

Sum-up of the week

- Introduction to ROOT, TMVA, SOFIE
- Build ROOT!
- Experiment with Python interface to SOFIE
- SOFIE Standalone, SOFIE Docker image
- Discussion on SOFIE Memory Allocation, Memory Planning
- Extend SOFIE coverage on user's models
 - SPANet
 - Smart Pixels
- SOFIE Dynamic Computation Support
- SOFIE Optimization modes
- Initial integration of ALPAKA into SOFIE- good discussions with Felice
- SOFIE interface with CLAD
- Unit test added to validate this functionality
- **Bug fixes!**

Interfacing Clad and SOFIE

- Worked on modifying SOFIE code emission to be compatible with Clad
 - Aim to convert trained NNs to headers
 - Wish to differentiate NN inference with Clad → use in likelihood evaluations (RooFit codegen)
- First, SOFIE refactored to emit code in Clad compatible way
 - Various merged improvements ([root#18297](#), [root#18324](#), [root#18341](#)), followed by open PR [root#18332](#)
- Then, implemented wrapper around calls of BLAS::sgemm_ ([root#18349](#))
- Finally, implement derivative of wrapper (pullback), for Clad to use, and add unit test of this ([root#18364](#))
- Should be ready to use in SBI project, providing significant speedup to likelihood evaluations/minimisations!
 - Basic (no Clad) SOFIE integration in workflow complete already
- With small additions, opens up the possibility of training models using emitted code

SOFIE ALPAKA Integration

- Using Kokkos Kernels for BLAS
- Initial implementation for NVIDIA GPUs (through Kokkos-Kernels)
- Currently supports GEMM and ReLU operations
- PR: <https://github.com/sanjibansg/SOFIE/pull/2>

```
#include "SOFIE/RModel.hxx"
#include "SOFIE/RModelParser_ONNX.hxx"
SOFIE::RModelParser_ONNX parser;
SOFIE::RModel model = parser.Parse("model.onnx");
model.GenerateGPU_ALPAKA();
model.OutputGenerated();
```

You, 1 second ago | 1 author (You)

//Code generated automatically by TMVA for GPU Inference using ALPAKA of Model file [Linear_16.onnx] at [Fri Apr 11 14:16:45 2025]

```
#ifndef SOFIE_LINEAR_16
#define SOFIE_LINEAR_16

#include <algorithm>
#include <vector>
#include <alpaka/alpaka.hpp>
#include <Kokkos_Core.hpp>
#include <KokkosBlas3_gemm.hpp>
#include "SOFIE/SOFIE_common.hxx"
#include <fstream>
```

```
using Dim1D = alpaka::DimInt<1>;
using Acc = alpaka::TagToAcc<alpaka::TagGpuCudaRt, Dim1D, Idx>;
using Queue = alpaka::Queue<Acc, alpaka::Blocking>;
```

You, 1 second ago | 1 author (You)

```
namespace SOFIE_Linear_16{
```

You, 1 second ago | 1 author (You)

```
struct Session {
```

```
    auto const platformHost = alpaka::PlatformCpu{};
    auto const devHost = alpaka::getDevByIdx(platformHost, 0);
    auto const platformAcc = alpaka::Platform<Acc>{};
    auto const devAcc = alpaka::getDevByIdx(platformAcc, 0);
    Queue queue(devAcc);
```

```
// initialized tensors
```

```
    auto deviceBuf_8weight = alpaka::allocBuf<float, size_t>(devAcc, 2500);
    auto deviceBuf_8bias = alpaka::allocBuf<float, size_t>(devAcc, 50);
    auto deviceBuf_4bias = alpaka::allocBuf<float, size_t>(devAcc, 50);
    auto deviceBuf_2weight = alpaka::allocBuf<float, size_t>(devAcc, 2500);
    auto deviceBuf_0bias = alpaka::allocBuf<float, size_t>(devAcc, 50);
    auto deviceBuf_12bias = alpaka::allocBuf<float, size_t>(devAcc, 50);
    auto deviceBuf_18bias = alpaka::allocBuf<float, size_t>(devAcc, 10);
    auto deviceBuf_14bias = alpaka::allocBuf<float, size_t>(devAcc, 50);
    auto deviceBuf_4weight = alpaka::allocBuf<float, size_t>(devAcc, 2500);
    auto deviceBuf_10weight = alpaka::allocBuf<float, size_t>(devAcc, 2500);
    auto deviceBuf_6bias = alpaka::allocBuf<float, size_t>(devAcc, 50);
    auto deviceBuf_18weight = alpaka::allocBuf<float, size_t>(devAcc, 500);
    auto deviceBuf_0weight = alpaka::allocBuf<float, size_t>(devAcc, 5000);
    auto deviceBuf_10bias = alpaka::allocBuf<float, size_t>(devAcc, 50);
```

```
//----- Gemm_GPU_ALPAKA
char op_0_transA = 'n';
char op_0_transB = 't';
int op_0_m = 16;
int op_0_n = 50;
int op_0_k = 100;
float op_0_alpha = 1;
float op_0_beta = 1;
int op_0_lda = 100;
int op_0_ldb = 100;
Kokkos::View<float**, Kokkos::LayoutLeft, Kokkos::CudaSpace> kokkos_dev_input1((float*)std::data(bufDev_input1), op_0_m, op_0_k);
Kokkos::View<float**, Kokkos::LayoutLeft, Kokkos::CudaSpace> kokkos_dev_0weight((float*)std::data(bufDev_0weight), op_0_k, op_0_n);
Kokkos::View<float**, Kokkos::LayoutLeft, Kokkos::CudaSpace> kokkos_dev_22((float*)std::data(bufDev_22), op_0_m, op_0_n);
KokkosBlas::gemm(&op_0_transB, &op_0_transA, op_0_alpha, kokkos_dev_input1, kokkos_dev_0weight, op_0_beta, kokkos_dev_22);
```

```
//----- Gemm_GPU_ALPAKA
char op_1_transA = 'n';
char op_1_transB = 't';
int op_1_m = 16;
int op_1_n = 50;
int op_1_k = 50;
float op_1_alpha = 1;
float op_1_beta = 1;
int op_1_lda = 50;
int op_1_ldb = 50;
Kokkos::View<float**, Kokkos::LayoutLeft, Kokkos::CudaSpace> kokkos_dev_22((float*)std::data(bufDev_22), op_1_m, op_1_k);
Kokkos::View<float**, Kokkos::LayoutLeft, Kokkos::CudaSpace> kokkos_dev_2weight((float*)std::data(bufDev_2weight), op_1_k, op_1_n);
Kokkos::View<float**, Kokkos::LayoutLeft, Kokkos::CudaSpace> kokkos_dev_24((float*)std::data(bufDev_24), op_1_m, op_1_n);
KokkosBlas::gemm(&op_1_transB, &op_1_transA, op_1_alpha, kokkos_dev_22, kokkos_dev_2weight, op_1_beta, kokkos_dev_24);
```

Future Plans

- Memory Allocator
 - Interface to allocate/free memory as SoA
 - Functionality to accept memory handlers from users
- Extend ALPAKA Integration
 - Improvements on data movement
 - Add implementation for more operators
 - Add interface to GPU accessors (to support other GPU stacks)
 - Add interface to BLAS libraries (to support other BLAS, ex. cuBLAS)
- Extend SYCL Integration
 - Replace portBLAS with Kokkos-Kernels

C++ Coroutines

Mateusz Jakub Fila, Eric Cano, Axel
Naumann, Attila Krasznahorkay



NexTGen
Next Generation Triggers

Sum up of the week

- C++20 introduced “coroutines”
 - <https://en.cppreference.com/w/cpp/language/coroutines>
- They are **super generic**
 - All aspects of how/what to resume, how the expectations are handled can be customised
 - They allow in-thread suspension and resume with a very low overhead
 - They are self contained and can be resumed() on different threads
 - Typically handled in libraries thread pool
 - Hopefully they could be used in our offline software for efficient asynchronous task scheduling. But also possibly other things.
- Mateusz' examples can be found in the repo:
 - <https://github.com/cern-nextgen/wp1.7-coroutine-tests>
- Toolkits/wrappers/abstraction layers also available:
 - <https://github.com/David-Haim/conurrencpp>
 - https://www.boost.org/doc/libs/1_88_0/libs/cobalt/doc/html/index.html

Sum up of the week

- Gaudi framework algorithms ported to coroutines
 - Most unit tests already pass
 - Some exception tests are still not passing
- Performance was measured
 - `resuming()` is, as expected, very lightweight (1.35 ns on a laptop, compared to 3.2 μ s for semaphore based)
- Tested functions and the plumbing with the bare bones awaitable classes



NextGen
Next Generation Triggers

